# FetchClimate Overview

version 1.2

FetchClimate is a combination of cloud-based service and a programmatic cloud that fetches necessary environmental data from multiple publicly available data sources. FetchClimate can be accessed from anywhere, either through a simple web interface, or via a few lines of code inside any .net program.

- any geographical region, at any grid resolution: from global, through continental, to a few kilometres
- any range of years (1900 – 2010), days within the year, and / or hours within the day
- wide range of environmental variables for both land and ocean
- automatic selection of the most appropriate source data
- uncertainty and provenance returned along with the climate information
- entire query can be shared as a single URL, enabling others to retrieve the identical data
- retrieval times vary – but are always orders of magnitude faster than doing this kind of thing the hard way!

This document covers what's necessary to run FetchClimate from your program or from OS command prompt.

## Contents


Getting started ..... 3

Command prompt queries ..... 3

Fetch climate from C++ ..... 3

Fetch climate from C#, Visual Basic, F# and other .NET languages ..... 4

FetchClimate C++API ..... 4

Single value ..... 4

Batch request ..... 4

Grid requests ..... 5

TimeSeries requests ..... 5

Other functions ..... 5

Functions parameters description ..... 5

Climate variables description ..... 8

Comments ..... 9

FetchClimate C# API (Procedural) ..... 9

Single region....10
Batch request....10
Grid requests....11
Time series requests....11
Missing value handling....12
Command line interface....12
General notes....13
Gridded interpolated station data data sources (WorldClim, CRU CL 2.0)....13
Time aggregated observations (GHCN)....14

# Getting started

FetchClimate runs on any Microsoft Windows computer which is connected to Internet and has Microsoft .NET Framework 4.0 installed. You will need only one file 'fetchclimate.exe' to run FetchClimate queries. The 'fetchclimate.h' file simplifies the use of the library in C++ code. This introductory section assumes you have Microsoft Visual Studio or Microsoft Visual Studio Express installed on your computer.

## *Command prompt queries*

The following command gets interpolated mean annual temperature at latitude=52.2°, longitude=0.1° (Cambridge, UK) for 30 years since 1961.

```
>fetchclimate t 52.2 0.1
Fetching FC_TEMPERATURE for 1 regions for whole days for whole years for years f
rom 1961 till 1990
Progress: 0%...100.0%
FetchClimate client version:    1.2.1193.0
FetchClimate service version:   1.2.875.0

Region #1 [point 52.2; 0.1]
9.717+-0.527 °C


>
```

To get this result he program passes parameters to FetchClimate service which does all the work. Invoke 'fetchclimate' without parameters to learn how to ask for specific climate data.

## *Fetch climate from C++*

You will need Microsoft Visual C++ compiler to run FetchClimate queries. Below is a minimum C++ program that prints interpolated mean annual temperature at latitude=52.2°, longitude=0.1° (Cambridge, UK) for 30 years since 1961.

```
#include <stdio.h>
#include "fetchclimate.h"

int main(int argc, char** argv[])
{
	double t = fetch_climate(FC_TEMPERATURE, 52.2, 52.2, 0.1, 0.1,
		FC_NOVALUE, FC_NOVALUE, FC_NOVALUE, FC_NOVALUE, 1961, 1990);
	printf("%lf", t);
	return 0;
}
```

The number being printed is 9.716667.

You have to enable compiler Common Language Runtime support (/clr option) and ensure that 'fetchclimate.exe' file is available both during compile and at run time. At compile time you may place 'fetchclimate'exe' in current directory. There are also several other options. Look into Visual C++ documentation for '#using' directive. At runtime the 'fetchclimate.exe' file must be at the same directory as the executable program itself, or it can be in any directory list in PATH environmental variable.

### *Fetch climate from C#, Visual Basic, F# and other .NET languages*

Reference 'fetchclimate.exe' assembly in your Microsoft Visual Studio project. The API is in 'Microsoft.Research.Science.Data' namespace. As an example, a C# program that prints interpolated mean annual temperature at latitude=52.2°, longitude=0.1° (Cambridge, UK) for 30 years since 1961 may look like this.

```
using System;
using Microsoft.Research.Science.Data;

namespace CS_Sample
{
    static void Main(string[] args)
    {
        double t = ClimateService.FetchClimate(ClimateParameter.FC_TEMPERATURE,
            52.2, 52.2, 0.1, 0.1);
        Console.WriteLine(t);
    }
}
```

Below is an equivalent F# script.

```
#r "FetchClimate.exe"
open Microsoft.Research.Science.Data
printf "%f\n" (ClimateService.FetchClimate(
                ClimateParameter.FC_TEMPERATURE,
                52.2,52.2,0.1,0.1,
                -999,-999,-999,-999,1961,1990,
                EnvironmentalDataSource.ANY))
```

## FetchClimate C++API

### *Single value*

```
double fetch_climate(int p, double latmin, double latmax, double lonmin, double lonmax, int
hourmin, int hourmax, int daymin, int daymax, int yearmin, int yearmax);
```

### *Batch request*

```
double* FetchClimateN(int p, double* latmin, double* latmax, double* lonmin, double* lonmax,
int* hourmin, int* hourmax, int* daymin, int* daymax, int* yearmin, int* yearmax, int N);
```

## Grid requests

```
void FetchClimateGrid(double* grid, int grid_dim1, int grid_dim2, int p, double latmin, double latmax, double lonmin, double lonmax, int hourmin, int hourmax, int daymin, int daymax, int yearmin, int yearmax, double dlat, double dlon);

double* FetchClimateGridEx(int p, double latmin, double latmax, double lonmin, double lonmax, int hourmin, int hourmax, int daymin, int daymax, int yearmin, int yearmax, double dlat, double dlon);
```

## TimeSeries requests

```
void FetchClimateYearly(double* result,int result_len,int p, double latmin, double latmax, double lonmin, double lonmax, int hourmin, int hourmax, int daymin, int daymax, int yearmin, yearmax, double dyear);

void FetchClimateSeasonly(double* result,int result_len,int p, double latmin, double latmax, double lonmin, double lonmax, int hourmin, int hourmax, int daymin, int daymax, int yearmin, int yearmax, double dday);

void FetchClimateHourly(double* result,int result_len,int p, double latmin, double latmax, double lonmin, double lonmax, int hourmin, int hourmax, int daymin, int daymax, int yearmin, int yearmax, double dhour);

double* FetchClimateYearlyEx(int p, double latmin, double latmax, double lonmin, double lonmax, int hourmin, int hourmax, int daymin, int daymax, int yearmin, yearmax, double dyear);
double* FetchClimateSeasonlyEx(p, double latmin, double latmax, double lonmin, double lonmax, int hourmin, int hourmax, int daymin, int daymax, int yearmin, int yearmax, double dday);

double* FetchClimateHourlyEx(p, double latmin, double latmax, double lonmin, double lonmax, int hourmin, int hourmax, int daymin, int daymax, int yearmin, int yearmax, double dhour);
```

## Other functions

```
bool is_missing_value(double value)
```

## Functions parameters description

| Paramenter | Description | Allowed values |
|---|---|---|
| *p* | An integer setting up what climate parameter needs to be fetched. It is usually set up with one of the predefined enum values. | FC_TEMPERATURE, FC_PRECIPITATION, FC_LAND_AIR_TEMPERATURE, FC_OCEAN_AIR_TEMPERATURE, FC_ELEVATION, FC_LAND_ELEVATION, FC_OCEAN_DEPTH, FC_SOIL_MOISTURE, FC_RELATIVE_HUMIDITY, FC_LAND_AIR_RELATIVE_HUMIDITY, FC_LAND_WIND_SPEED, |

| Paramenter | Description | Allowed values |
|---|---|---|
| | | `FC_LAND_DIURNAL_TEMPERATURE_RANGE`, `FC_LAND_FROST_DAY_FREQUENCY`, `FC_LAND_WET_DAY_FREQUENCY`, `FC_LAND_SUN_PERCENTAGE` |
| *Latmin* | The bottom bound value in degrees of requested spatial rectangle for calculating mean climate parameter value. | -90 .. -90 inclusive |
| *Latmax* | The top bound value in degrees of requested spatial rectangle for calculating mean climate parameter value. | -90 .. 90 inclusive |
| *Lonmin* | The left bound value in degrees of requested spatial rectangle for calculating mean climate parameter value. | -180..180 or 0..360<br>*Lonmin* and *Lonmax* must follow the same notation. You must use either -180..180 or 0.360 notation in both *Lonmin* and *Lonmax* |
| *Lonmax* | The right bound value in degrees of requested spatial rectangle for calculating mean climate parameter value. | -180..180 or 0..360<br>*Lonmin* and Lonmax must follow the same notation. You must use either -180..180 or 0.360 notation in both *Lonmin* and *Lonmax* |
| *dLat* | a step of latitude splitting that will be used to produce grid. The first dimension of the output grid that corresponds to latitude will have a length equal to (latmax-latmin)/dlat. Latmax value will be corrected to keep (latmax-latmin)/dlat as integer value. | more than 0 |
| *dLon* | a step of longitude splitting that will be used to produce grid. The second dimension of the output grid that corresponds to longitude will have a length equal to (lonmax-lonmin)/dlon. Latmax value will be corrected to keep (lonmax-lonmin)/dlon as integer value | more than 0 |

| Paramenter | Description | Allowed values |
|---|---|---|
| *Hourmin* | A lower inclusive bound of time interval in a day that will be used for calculation. It can be set to FC_NOVALUE that will be treated as 0. | 0..24 (or FC_NOVALUE in C++) |
| *hourmax* | A higher inclusive bound of time interval in a day that will be used for calculation. It can be set to FC_NOVALUE that will be treated as 24. | 0..24 (or FC_NOVALUE in C++) |
| *Daymin* | A number of the day in the year that will be the first day within each year of the requested time period. It can be set to FC_NOVALUE that will be treated as the 1th of January. | 1..365(366) (or FC_NOVALUE in C++)<br>if query covers a single year, the maximum value of *Daymin* can be 365 if the year is not leap or 366 if the year is leap.<br>If query covers several years, it is assumed that all years are not leap and maximum value of *Daymin* must be less or equal to 365 |
| *Daymax* | A number of the day in the year that will be the last day within each year of the requested time period. It can be set to FC_NOVALUE that will be treated as the 31th of December. | 1..365(366) (or FC_NOVALUE in C++)<br>if query covers a single year, the maximum value of *Daymax* can be 365 if the year is not leap or 366 if the year is leap.<br>If query covers several years, it is assumed that all years are not leap and maximum value of *Daymax* must be less or equal to 365 |
| *yearmin* | The number of year that will be the first year in the requested period | 1 .. 9999 |
| *yearmax* | The number of year that will be the last year in the requested period | 1 .. 9999 |
| *grid* | A pointer to the preallocated array where the result data will be put in the grid request functions. | |
| *result* | A pointer to the preallocated array where the result data will be put in the time series request functions | |
| *grid_dim1* | The length of the first dimension of preallocated array where the result | 0 .. n |

| Paramenter | Description | Allowed values |
|---|---|---|
| | will be put in the grid request functions. | |
| *grid_dim2* | The length of the second dimension of preallocated array where the result will be put in the grid request functions. | 0 .. n |
| *result_len* | The length of the preallocated array where the result will be put in the time series request functions. | 0 .. n |
| *N* | The length of all arrays that are used as a arrays with parameters in function FetchClimateN | 0 .. n |
| *dyear* | A value that divides [yearmin, yearmax] interval into periods of dyear length to form output grid. | 1 .. n |
| *dday* | A value that divides [daymin, daymax] interval into periods of dday length to form output grid. | 1 .. n |
| *dhour* | A value that divides [hourmin, hourmax] interval into periods of dhour length to form output grid. | 1 .. n |

## *Climate variables description*

FetchClimate supports different coverage types.

- Only land area in the requested spatial region (L).
- Only ocean area in the requested spatial region (O).
- Both land and ocean (W).

The following table describes the mapping of FetchClimate API parameter to corresponding climate parameter and coverage type

| Fetch Climate API Parameter | Climate parameter | Coverage type | Default units |
|---|---|---|---|
| FC_TEMPERATURE | Near surface air temperature | (W) | Degrees Celsius |
| FC_LAND_AIR_TEMPERATURE | Near surface air temperature | (L) | Degrees Celsius |

| Fetch Climate API Parameter | Climate parameter | Coverage type | Default units |
|---|---|---|---|
| FC_OCEAN_AIR_TEMPERATURE | Near surface air temperature | (O) | Degrees Celsius |
| FC_PRECIPITATION | Near surface precipitation rate | (W) | mm/month |
| FC_SOIL_MOISTURE | Soil moisture (upper soil layers) | (L) | mm/m |
| FC_ELEVATION | Earth surface elevation (land and ocean) | (W) | meters |
| FC_LAND_ELEVATION | Land elevation above sea level | (L) | meters |
| FC_OCEAN_DEPTH | Bathymetry | (O) | meters |
| FC_LAND_AIR_RELATIVE_HUMIDITY | Air related humidity near surface | (L) | percentage |
| FC_LAND_WIND_SPEED | Wind speed near surface | (L) | Meters/second |
| FC_LAND_DIURNAL_TEMPERATURE_RANGE | Diurnal temperature range | (L) | Degrees Celsius |
| FC_LAND_FROST_DAY_FREQUENCY | Number of days with ground frost in month | (L) | Days per month |
| FC_LAND_WET_DAY_FREQUENCY | Number of days with precipitation in month | (L) | Days per month |
| FC_LAND_SUN_PERCENTAGE | Sunshine percentage of maximum possible sunshine during a day | (L) | Percentage |

### *Comments*

If pair (hourmin, hourmax) is set to (FC_NOVALUE, FC_NOVALUE) or (0, 24) then the data for whole day will be used in calculation.

If pair (daymin, daymax) is set to (FC_NOVALUE, FC_NOVALUE) then the whole year will be used in calculation

# FetchClimate C# API (Procedural)

To use FetchClimate C# API we need to create .NET 4.0 project and add a reference to **FetchClimate.exe** assembly. Procedural API consists of static methods of **ClimateService** class of `Microsoft.Research.Science.Data` namespace.

All methods require only climate variable specification and region specification as mandatory parameters. Time interval specification is optional. If not specified it is considered to be a continuous period from 1961 to 1990.

Time interval can be refined by modifying default values of the following parameters: starthour, stophour, startday,stopday,startyear,stopyear.

As well as time interval we can specify a data source to use for calculation through an optional **dataSource** parameters. The default parameter value is **EnvironmentalDataSource.ANY** means that most accurate data source should be chosen automatically.

With a FetchClimate C# procedural API it is possible to fetch uncertainties and provenance of the mean values as well as mean values themselves. Methods for fetching uncertainties and provenance are described below.

## *Single region*

Fetching a mean value of specified environmental variable for one region

```
public static double FetchClimate(ClimateParameter parameter, double latmin, double latmax,
double lonmin, double lonmax, int starthour=0, int stophour=24, int startday=1, int
stopday=GlobalConsts.DefaultValue, int startyear=1961, int stopyear =
1990,EnvironmentalDataSource dataSource=EnvironmentalDataSource.ANY)
```

Fetching an estimated uncertainty for single region request.

```
public static double FetchClimateUncertainty(ClimateParameter parameter, double latmin, double
latmax, double lonmin, double lonmax, int starthour = 0, int stophour = 24, int startday = 1,
int stopday = GlobalConsts.DefaultValue, int startyear = 1961, int stopyear = 1990,
EnvironmentalDataSource dataSource = EnvironmentalDataSource.ANY)
```

Fetching a result provenance for single region request

```
public static string FetchClimateProvenance(ClimateParameter parameter, double latmin, double
latmax, double lonmin, double lonmax, int starthour = 0, int stophour = 24, int startday = 1,
int stopday = GlobalConsts.DefaultValue, int startyear = 1961, int stopyear = 1990,
EnvironmentalDataSource dataSource = EnvironmentalDataSource.ANY)
```

## *Batch request*

Perform a batch request: fetch mean values of an environmental variable for several regions specifying time intervals for each particular region.

```
public static double[] FetchClimate(ClimateParameter parameter, double[] latmin, double[]
latmax, double[] lonmin, double[] lonmax, int[] starthour = null, int[] stophour = null, int[]
startday = null, int[] stopday = null, int[] startyear = null, int[] stopyear = null,
EnvironmentalDataSource dataSource = EnvironmentalDataSource.ANY)
```

Fetch estimated uncertainties in a batch request.

```
public static double[] FetchClimateUncertainty(ClimateParameter parameter, double[] latmin,
double[] latmax, double[] lonmin, double[] lonmax, int[] starthour = null, int[] stophour =
```

```
null, int[] startday = null, int[] stopday = null, int[] startyear = null, int[] stopyear =
null, EnvironmentalDataSource dataSource = EnvironmentalDataSource.ANY)
```

Fetch data sources in a batch request.

```
public static string[] FetchClimateProvenance(ClimateParameter parameter, double[] latmin,
double[] latmax, double[] lonmin, double[] lonmax, int[] starthour = null, int[] stophour =
null, int[] startday = null, int[] stopday = null, int[] startyear = null, int[] stopyear =
null, EnvironmentalDataSource dataSource = EnvironmentalDataSource.ANY)
```

## *Grid requests*

With a grid request it is possible to fetch mean values of some environmental variable for the cells of a grid specified. Each value of the resulting array is a mean value for corresponding grid cell. The Grid is defined by its spatial bounds, by a step along latitude and by a step along longitude.

Fetching mean values of a climate variable.

```
public static double[,] FetchClimateGrid(ClimateParameter parameter, double latmin, double
latmax, double lonmin, double lonmax, double dlat, double dlon, int hourmin = 0, int hourmax =
24, int daymin = 1, int daymax = GlobalConsts.DefaultValue, int yearmin = 1961, int yearmax =
1990, EnvironmentalDataSource dataSource = EnvironmentalDataSource.ANY)
```

Fetching an uncertainties array for grid request values.

```
public static double[,] FetchUncertaintyGrid(ClimateParameter parameter, double latmin, double
latmax, double lonmin, double lonmax, double dlat, double dlon, int hourmin = 0, int hourmax =
24, int daymin = 1, int daymax = GlobalConsts.DefaultValue, int yearmin = 1961, int yearmax =
1990, EnvironmentalDataSource dataSource = EnvironmentalDataSource.ANY)
```

Fetching a provenance array for grid request values.

```
public static string[,] FetchProvenanceGrid(ClimateParameter parameter, double latmin, double
latmax, double lonmin, double lonmax, double dlat, double dlon, int hourmin = 0, int hourmax =
24, int daymin = 1, int daymax = GlobalConsts.DefaultValue, int yearmin = 1961, int yearmax =
1990, EnvironmentalDataSource dataSource = EnvironmentalDataSource.ANY)
```

## *Time series requests*

Time series requests give an ability to split single request into several requests either along years, or along days within year or along hours within day. **FetchClimateYearly** splits the years interval with a step specified in dyear paramerer. **FetchClimateSeasonly** splits subinterval of days within each year into several subintervals with a step of dday. **FetchClimateHourly** splits subinterval of hours within each day into several subintrervals with a step of dhours. If the interval to split is not multiple of a step the later bound of last subinterval can be trimmed not to exceed the later bound.

```
public static double[] FetchClimateYearly(ClimateParameter p, double latmin, double latmax,
double lonmin, double lonmax, int hourmin = 0, int hourmax = 24, int daymin = 1, int daymax =
```

```
GlobalConsts.DefaultValue, int yearmin = 1961, int yearmax = 1990, int dyear = 1,
EnvironmentalDataSource dataSource = EnvironmentalDataSource.ANY)

public static double[] FetchClimateSeasonly(ClimateParameter p, double latmin, double latmax,
double lonmin, double lonmax, int hourmin = 0, int hourmax = 24, int daymin = 1, int daymax =
GlobalConsts.DefaultValue, int yearmin = 1961, int yearmax = 1990, int dday = 1,
EnvironmentalDataSource dataSource = EnvironmentalDataSource.ANY)

public static double[] FetchClimateHourly(ClimateParameter p, double latmin, double latmax,
double lonmin, double lonmax, int hourmin = 0, int hourmax = 24, int daymin = 1, int daymax =
GlobalConsts.DefaultValue, int yearmin = 1961, int yearmax = 1990, int dhour = 1,
EnvironmentalDataSource dataSource = EnvironmentalDataSource.ANY)
```

Get corresponing uncertatinty and provenance values the following methods can be used.

For yearly timeseries: **FetchClimateYearlyUncertainty, FetchClimateYearlyProvenance.**

For seasonly timeseries: **FetchClimateSeasonlyUncertatinty, FetchClimateSeasonlyProvenance.**

For hourly timeseries: **FetchClimateHourlyUncertainty, FetchClimateHourlyProvenance** .

### *Missing value handling*

Missing value is supposed to appear in case of fetching coverage specific (i.e. only land without an ocean) parameter for area with different coverage. For instance fetching over land air temperature for ocean area will return missing value. In C API the values returned by the service should be checked by is_missing_value function. In C# API the values containing missing values equal to double.NAN.

## Command line interface

**FetchClimate.exe** can also be used from a command prompt or a batch file. Command line parameters specify which data to fetch and where the result will go. At a minimum you have to specify climate variable and one geographical region. Default time interval is set to continuous years from 1961 to 1990 and can be overridden. When several spatial regions are specified on a command line, one climate variable and time interval are applied to all requested spatial regions. By default, FetchClimate.exe prints the formatted result to standard output. The result can also be exported either to plain CSV file or to annotated CSV (readable by SDS as a dataset). The below command line syntax is printed if you invoke FetchClimate.exe without parameters.

```
FetchClimate /clearcache
FetchClimate <parameter> [<TimeSpec>] <RegionSpec> {<RegionSpec>} [/export plain
|annotated FILENAME] [/dsv]

<RegionSpec>    :=  <PointSpec>|<RectSpec>|<GridSpec>
<PointSpec>     :=  <lat> <lon>
<RectSpec>      :=  <latMin>:<latMax> <lonmMin>:<lonMax>
<GridSpec>      :=  <latMin>:<latStep>:<latMax> <lonMin>:<lonStep>:<lonMax>
```

```
<TimeSpec>      :=  [/years <yearmin> <yearmax>] [/days <daymin> <daymax>] [/hou
rs <hourmin> <hourmax>] [<TimeSeriesSpec>]
<TimeSeriesSpec>      :=  /ty <YearStep> | /ts <DayStep> | /th <HourStep>

<parameter>     :=  t|FC_TEMPERATURE|p|FC_PRECIPITATION|lt|FC_LAND_AIR_TEMPERATU
RE|ot|FC_OCEAN_AIR_TEMPERATURE|e|FC_ELEVATION|le|FC_LAND_ELEVATION|od|FC_OCEAN_D
EPTH|sm|FC_SOIL_MOISTURE|rh|FC_RELATIVE_HUMIDITY|lrh|FC_LAND_AIR_RELATIVE_HUMIDI
TY|lwnd|FC_LAND_WIND_SPEED|ldtr|FC_LAND_DIURNAL_TEMPERATURE_RANGE|lfdf|FC_LAND_F
ROST_DAY_FREQUENCY|lwdf|FC_LAND_WET_DAY_FREQUENCY|lsp|FC_LAND_SUN_PERCENTAGE

/clearecache        -   clear client-side cache
/ty <YearStep>      -   perform yearly timeseries request by dividing specified
years into <YearStep> long subintervals
/ts <DayStep>       -   perform seasonly timeseries request by dividing specifie
d days into <DaysStep> long subintervals
/th <HourStep>      -   perform hourly timeseries request by dividing specified
hours into <HourStep> long subintervals
/and                -   add an additional region to fetch

If <TimeSpec> is omitted  the mean value for whole years from 1961 to 1990 is fetched.
```

# General notes

Service works with 8 data sources. They are NCEP Reanalysis 1, CRU CL 2.0, WorldClim 30s, GHCN v2, CPC Soil Moisture, GTOPO30 and ETOPO1

All time is now interpreted as UTC time. If you specify "morning" hours in your request while requesting a wide spatial area, the result that will be returned won't correspond to average morning time for this area. It will contain an average for specified UTC hours across whole region.

## *Gridded interpolated station data data sources (WorldClim, CRU CL 2.0)*

Data sources with Gridded interpolated station data like WorldClim or CRU will not calculate a mean value for a region containing missing value (e.g water area), some other less accurate data source will be used to get a value for this type of regions.

If the requested days of year do not fit month bound, monthly average values will be weighted according to the length of submonth period.

For example, in the case of requesting mean temperature for days from the 10th of June to the 10th of August, the result will be generated using mean temperature of June,July and August with corresponding weights 0.7, 1 and 0.32.

Weight of June= requested days in June / total days in June = 21/30 =0.7

Weight of July= requested days in July / total days in July = 31/31 =1

Weight of August = requested days in August / total days in August = 10/31 =0.32

The resulting mean value will be the following:

(JuneTemp*0.7+JulyTemp*1 + AugustTemp*0.32)/(0.7+1+0.32)

## *Time aggregated observations (GHCN)*

GHCN Data source uses only those stations to get a result for the request, that contain continuous timeseries (e.g. all requested months for all requested years are present in the data). If at least one of the requested months of any of the requested years is absent for some station in the requested spatial region, the station data will be ignored.

Major effect for precision calculation for GHCN data source is caused by station density in a region. So it is likely that GHCN data source will only be used in the high station density regions (e.g. USA) or if other data sources fail their calculations for the particular request.